REMARKS/ARGUMENTS

This amendment is submitted in response to the Final Office Action dated June, 6, 2006 and is being submitted with a Request for Continued Examination (RCE). Claims 1-69 are pending in the present application. Claims 1-55 and 69 have been amended. No claims have been canceled or added. Claims 1-69 remain pending.

Claim 1 has been amended to recite "wherein an execution path of the non-critical portions is visible to a debugger program executing in the user-level protected mode" and "wherein an execution path of the critical code segments is hidden from the debugger program executing in the user-level protected mode". Independent claims 15, 16, 36, 55 and 69 have been amended to include similar recitations. Support for the amendments can be found throughout the specification. See for example page 8, lines 7-22, page 10, page 12, page 14, page 16, and page 18.

Claims 1-54 were amended to cancel unnecessary reference to "the step of" and "the steps of". These amendments are seen by Applicant as broadening or cosmetic, and as such, is not subject to the prosecution history estoppel imposed by Festo. For the record, Applicant points out that the Supreme Court in Festo noted that a cosmetic amendment would not narrow the patent's scope and thus would not raise the estoppel bar.

The Examiner rejected claims 1-7, 15-18, 26, 36-38, 55-61 and 69 under 35 USC §103(a) as being unpatentable over Kuzara et al. (US 5,450,586) in view of Held (US 5,889,988). The Examiner rejected claims 8-14, 19-25, 27, 39-54, 62-68 under 35 USC §103(a) as being unpatentable over Kuzara in view of Held and further in view of Cardoza et al. (US 5,630,049). Claims 28-31 are rejected under 35 USC §103(a) as

being unpatentable over Kuzara in view of Held and further in view of Admitted Prior Art (APA) and Hagimont et al. "Hidden Software Capabilities". Claims 32-35 are rejected under 35 U.S.C. §103(a) as being unpatentable over Kuzara, Held, and Hagimont (in light of APA) and further in view of Cardoza. Applicants respectfully disagree.

The present invention is a method and system for increasing software program security by obfuscation of program execution flow. The obfuscation of program execution flow hides key algorithms from view during debugging to increase the difficulty of reverse engineering, and to increase the difficulty of determining how to defeat anti-piracy features in the software. This is accomplished by hiding the execution flow of selected or critical portions of the program to be protected from a software debugger program when the debugger is run in an operating system that supports dual mode operation. i.e., user-mode (protected mode) and kernel-mode (unprotected mode).

In contrast, the references cited by the Examiner disclose methods for analyzing and debugging software. None of the references alone or in combination teach or suggest the combination of "executing non-critical portions of the software program in the user-level protected mode, wherein an execution path of the non-critical portions is visible to a debugger program executing in the user-level protected mode" and "executing the critical code segments within respective exception handlers in the kernel-level unprotected mode, wherein an execution path of the critical code segments is hidden from the debugger program executing in the user-level protected mode" as recited in amended claim 1.

As argued in the previous Amendment, Kuzara discloses a system for debugging a software system in which code markers are inserted by the user at compile time or interactively during a debug session to make visible critical points in code execution. (See Summary); and Held discloses a debugger that provides a system such that an end user can debug drivers of the operating system itself while still using a graphical user interface (Summary).

As stated on page 6, line 13+ of the Specification, a debugger is used to study the execution flow of a program, set software breakpoints, and analyze the contents of the microprocessor registers, data structures, and program variables at various points in the execution of a program. This process is normally associated with locating and correcting program errors during the development of a software program, and is referred to as "debugging" the software program. However, the same tool can be used to analyze the operation of any program, and thus is often used by hackers to determine the operation of a program they desire to make freely available. It can also be used to reverse-engineer software by helping an engineer extract key algorithms from a software program.

As state on page 9, line 21+, according to the present invention, the security of the software program is increased by obfuscating the execution flow of the program when the software program is executed on the computer system and analyzed by the debugger program. The execution flow of the software program is obfuscated by identifying critical code segments in the software program that need to be hidden, and then executing the non-critical portions of the software program in the user-level protected mode, as normal, while the critical code segments are executed within

respective exception handlers in the kernel-level unprotected mode. User-level debuggers are incapable of analyzing code executed within exception handlers kernel-level mode and will therefore be unable to provide information regarding the critical code segments of the Software program for a hacker to reverse engineer.

Not only do the references fails to teach or suggest the claims of the present invention, the references actually teach away from the claims of the present invention but because the references teach methods for making debugging code more visible, which makes debugging code easier, rather than hiding of execution of critical codes segments from the user-level debugger.

The Examiner cited Col. 5, lines 30-49 of Kazura for teaching the hiding of execution of critical codes segments from a debugger program. However, this merely states:

> The first aspect of the present invention comprises means for inserting code markers directly into the embedded system under investigation at compile time or indirectly by inserting the code markers into the interface library of the RTOS service calls. As used herein, the RTOS service call library is the interface layer between the user's application code and the operating system "black box". By calling the RTOS service call library when calling the RTOS, these code markers may be inserted without the user knowing the inner workings of the operating system. This is accomplished by writing out information at all entry and exit points into and from the RTOS "black box", thereby making the RTOS activity visible without having to know the inner workings of the specific RTOS. The information is then written out to a specific, contiguous marker data area, and the written information identifies not only which RTOS service call was called but also the input parameters, the output parameters and the return values to/from the RTOS.

Thus, this passage teaches inserting code markers to make RTOS activity visible without the user having to know the inner workings of the RTOS, which is effectively

hiding the operating system from the user, rather than hiding critical code segments from a debugger.

Held and Cardoza fail to cure this deficiency of Kuzara. Thus, Kuzara in combination with Held and Cardoza fail to teach or suggest the above mentioned combination of elements, as recited in amended claim 1. Thus, claim 1 is allowable over the references.

Amended independent claims 15, 16, 36, 55 and 69 recite similar and/or additional limitations and are also allowable over the references for at least the same reasons as amended claim 1. Moreover, the remaining claims, which depend either directly or indirectly from one of claims 1, 15, 16, 36, 55 and 69, are considered allowable for at least these same reasons.

In view of the foregoing, it is submitted that claims 1-69 are allowable over the cited references. Accordingly, Applicant respectfully requests reconsideration and passage to issue of claims 1-69 as now presented. Should any unresolved issues remain, the Examiner is invited to call Applicants' attorney at the telephone number indicated below.

Respectfully submitted,
STRATEGIC LAW GROUP

August 9, 2006
Date

/Stephen G. Sullivan/
Stephen G. Sullivan
Attorney for Applicant(s)
Reg. No. 38,329
(650) 969-7474